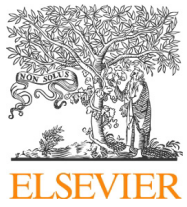

Procedia Computer Science

Volume 29, 2014, Pages 499–508

ICCS 2014. 14th International Conference on Computational Science

A Workflow Application for Parallel Processing of Big Data from an Internet Portal*

Paweł Czarnul

Faculty of Electronics, Telecommunications and Informatics
Gdansk University of Technology, Poland
pczarnul@eti.pg.gda.pl

Abstract

The paper presents a workflow application for efficient parallel processing of data downloaded from an Internet portal. The workflow partitions input files into subdirectories which are further split for parallel processing by services installed on distinct computer nodes. This way, analysis of the first ready subdirectories can start fast and is handled by services implemented as parallel multithreaded applications using multiple cores of modern CPUs. The goal is to assess achievable speed-ups and determine which factors influence scalability and to what degree. Data processing services were implemented for assessment of context (positive or negative) in which the given keyword appears in a document. The testbed application used these services to determine how a particular brand was recognized by either authors of articles or readers in comments in a specific Internet portal focused on new technologies. Obtained execution times as well as speed-ups are presented for data sets of various sizes along with discussion on how factors such as load imbalance and memory/disk bottlenecks limit performance.

Keywords: parallel data processing, parallel performance, Internet, parallel workflow application

1 Introduction

Recent developments in hardware, from multicore processors and GPU cards up to clusters of nodes equipped with multiple CPUs and GPUs have made it possible to speed up computations considerably. At the same time, cheap and faster and faster disk storage solutions including SSD disks allow reading and writing large amounts of data fast. This allows downloading parts of the huge data stored in the Internet for further off-line analysis. This has many applications starting from analysis of tendencies in stock values up to assessment of brand recognition. Consequently, maximizing speed-up and identification of performance bottlenecks especially for large data sizes is crucial in many fields. The latter is the problem addressed in this work.

The outline of the paper is as follows. Section 2 discusses related work and motivations. Section 3 presents the design of a workflow application for parallel analysis of data. Section 4 presents exper-

*work performed within grant supported from Polish National Science Center based on decision DEC-2012/07/B/ST6/01516

doi:10.1016/j.procs.2014.05.045

iments with execution times and speed-ups and discussion on the factors limiting the latter. Finally, Section 5 discusses planned future work extending the presented results.

2 Related Work and Motivations

In data parallel processing data is partitioned into chunks that are then analyzed in parallel by either processes or threads. This includes embarrassingly parallel computations [3] or geometric computations with problems in areas such as electromagnetics, computational fluid dynamics [3], medicine [9].

2.1 Parallel Data Processing Frameworks

The authors of [10] have demonstrated efficient usage of the MapReduce scheme for large data processing on a cluster of machines. They demonstrated how several applications could be implemented in the framework including: distributed grep for text matching, count of URL visits from server logs, web link graph, searching for most important words in documents, inverted index and distributed sort. The solution is able to cope with machine failures. The paper presents results for a collection of over 1700 hosts. The most relevant to this work is the distributed grep in which the input is split into 15000 pieces each 64 MB in size and the output reduced to 1 file. The entire computations takes approximately 150 seconds with overhead for program propagation and delays caused by the GFS system.

Recently, parallel data processing in modern distributed environments has gained attention. For instance, paper [18] presents a research project Nephele for data parallel processing in IaaS clouds which is compared to Hadoop. An experiment is set up that demonstrates the advantage of Nephele to use dynamic resource provisioning in compute clouds that allows it to gain an advantage over Hadoop. The application sorts and selects integer numbers. Nephele allows modeling application as a workflow graph (DAG) and specify parallelization levels.

Technical aspects and optimizations of MapReduce for parallel data processing are discussed in [13]. Paper [11] presents a solution for massive spatial data processing on high performance computing cluster. The latter demands heavy I/O operations. Large data sets could also be partitioned and processed in parallel using volunteer based systems such as BOINC [1] or COMCUTE [2].

2.2 Parallel Text Processing Applications

From the application point of view, of particular interest in this work is processing of data that can be gathered from the Internet, in the context of information retrieval. The latter is the process of identification and obtaining relevant documents based on a query [12]. There are approaches for parallel handling of queries on a multiprocessor system such as [5] where the speed-up of 11.3 is obtained for 16 processors. Various approaches are also discussed in [4]. There are several methods available for parallel text search [16]. Paper [15] presents a text search mechanism on a low cost cluster with a performance model and verification of the latter against real experiments. The factor that influences performance is load balancing. Work [14] shows performance evaluation of parallel information retrieval on a multiprocessor system with consideration of the number of CPUs, threads, disks etc. However, the number of CPUs and threads are limited to 4 and 32 respectively.

There are several applications run in this context regularly such as assessment of market values, brand recognition, processing of stock values etc. Correct assessment of context is needed for drawing viable conclusions and advantage over other search methods [17]. For subsequent analysis of data, an index will be built to speed up subsequent searches [4]. On the other hand, analysis of new data e.g. new articles that show up on a daily basis or readers' comments might be analyzed just once after download e.g. for determination of tendencies from day to day and month to month.

2.3 Motivations and Contributions

In view of the aforementioned works, the motivations of this work are all of the following:

1. Assessment of achievable speed-up levels (including HyperThreading technology) and the factors limiting the speed-up (along with impact levels) in a parallel cluster-based system when processing typical data fetched from Internet portals. The considered factors limiting the speed-up are especially the impact of access to shared storage space and potential load imbalance.

2. Design a reusable and scalable workflow application for parallel processing of data fetched from an Internet portal that would be suitable for both parallel and distributed systems. It includes deployment of the workflow application in a workflow management system of BeesyCluster for combination of benefits offered by the latter in one system: fault-tolerant processing in case of service failures [8] similarly to [10], dynamic service/resource selection [8] similarly to Nephele [18] and consideration of storage constraints for workflow nodes [7]. Compared to [15] and [14], larger configurations are considered. Even though the focus of the paper is on performance, this combination of features make it functionally a very rich solution compared to the others.

3 Design of the Solution

3.1 Workflow Modeling

In order to achieve efficient parallel processing of big data, the author proposes a workflow application depicted in Figure 1. The workflow considers the following steps, assuming the data is already in a designated location/space, available as a collection of files.

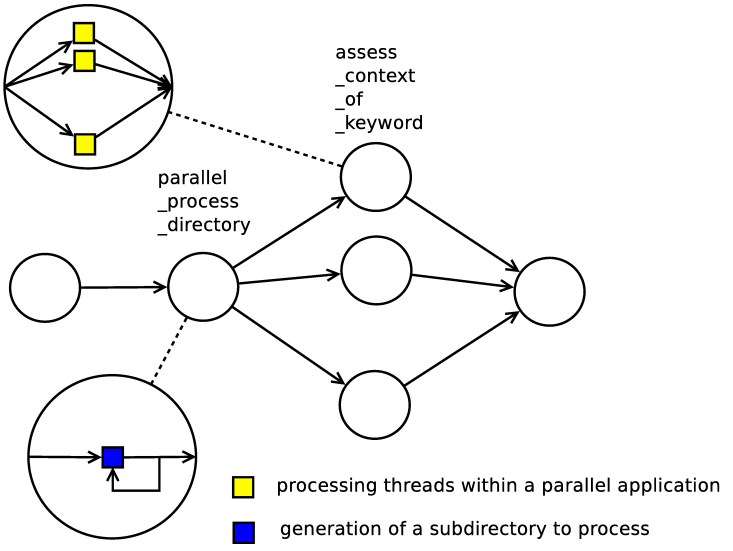

Figure 1: A workflow application for parallel processing of data

Service `parallel_process_directory` splits the initial directory into reasonably large subdirectories, partitions input files in a successive subdirectory and initializes parallel computations by services `assess_context_of_keyword`.

Service `assess_context_of_keyword` assesses the context of the given keyword in a list of files in parallel. The application, written in C using the Pthreads library, reads file names assigned to the application and partitions into arrays to be assigned to particular threads. Each thread processes files to detect existence of the given keyword and then positive and negative descriptive words. If the keyword is present in the file (which can be an article in a portal, for example), the context of `keyword` within file (document, article) a is evaluated as $f_c(keyword, a) = e_p(a) - e_n(a)$ where $e_p(a)$ is the number of positive descriptive words in a while

$e_n(a)$ is the number of negative descriptive words in a. If the keyword is present in a, then $f_c(keyword)$ is updated with $f_c(keyword) + f_c(keyword, a)$ i.e. the computed value is added to the total value of keyword across all articles. This is stored in a separate file.

One can notice that there are two partitioning issues: across computer nodes and across cores within a single node. Assuming same processing speeds of nodes, the partitioning problem is minimization of the load across the partitions which depends linearly on the file sizes assigned to the partitions. The problem can be stated as:

$$\forall_{f_i \in I} \exists_{F_j} f_i \in F_j; \quad \text{minimize } max_j \sum_{f_i \in F_j} (\alpha + s_i) \tag{1}$$

where I denotes the set of input files to process, s_i denotes the size of file i, F_j denotes the set of files assigned to partition j (assigned to a thread) and α is a constant corresponding to the overhead of file opening, closing etc. Each input file must be assigned to exactly one partition. This problem is difficult assuming many files of different sizes. Good partitioning algorithms might take a long time which may be a factor limiting speed-up given that data analysis is reasonably fast. For this reason, as described in Section 4.3, partitioning based on the number of files was used. This scheme allows for more coarse load balancing but minimizes the time of the load balancing phase. It is based on the assumption, that given a large number of files (like in Internet portals), average sizes assigned to various partitions will be similar.

It should be noted that the proposed workflow application does not consider the acquisition phase which precedes processing. While it is out of scope of this work, it should be noted that the acquisition phase could also be parallelized:

1. fragments of the portal can be fetched independently by processes running on different nodes,

2. upon receiving a part of the data, it can be streamed to the following node for processing while remaining data is being fetched in parallel.

For actual deployment of the workflow, the author proposed to use the Workflow Management System of BeesyCluster [8, 6] that allows modeling of parallel paths and assignment of both downloading as well as processing services which is demonstrated in Figure 2 for the proposed workflow application used in subsequent tests as described in Section 4. Services `parallel_process_directory` and `assess_context_of_keyword` corresponding to the applications described above are assigned to workflow nodes as in Figure 2. BeesyCluster has been designed in such a way that in case of service failure, an alternative service for the task is executed or the same service can be rerun. If there are more than one service assigned per task e.g. processing data on various speed/monetary cost conditions, the workflow management subsystem can optimize selection of services per tasks taking into account QoS parameters. A global goal can be optimized such as minimization of the workflow execution time with a bound on the total cost of the selected services [8]. BeesyCluster can obey storage limitations of the locations where services are installed [7]. This combination of features makes it a very flexible solution compared to the ones analyzed in Section 2.1.

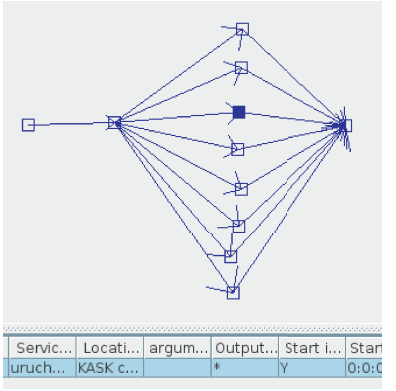

Figure 2: Workflow application in BeesyCluster Workflow Management System

3.2 Proposed Performance Optimizations

3.2.1 Minimization of Initial Latency

In case there are many files in the input data set, e.g. greater than 20000 the partitioning time of the input data set may take several seconds. In order to mitigate this issue, the partitioning phase is preceded by short and fast partitioning of the input data set into several disjoint sets. Subsequently, the smaller sets are processed one by one. This solution offers the following benefits:

1. the smaller data gets partitioned faster and reaches the processing nodes faster,
2. partitioning of successive directories and initialization of computations by services `assess_context_of_keyword`. In this context, this allows pipelined computations in which computations are initialized and next directories can be preprocessed for generation of files to be assigned to various computer nodes.

3.2.2 Hiding Communication Latency

The application uses the given number of nodes and the predefined number of cores per node. Within each node, threads are launched in order to accommodate the cores. The thread has the operational cycle in which it subsequently reads data from the file and processes it. The application has the capability to launch more threads per core e.g. 2 threads per core in order to fully utilize the core while the other is waiting for I/O. However, this introduces overhead for creation and switching between the threads.

4 Experiments

4.1 Testbed Workflow Application

The author has used the workflow scheme proposed in Section 3 and implemented service `assess_context_of_keyword` as a parallel application for assessment whether a particular keyword appears in either positive or negative context in an article in an Internet portal.

Each service was developed as an application coded in C using Pthreads for multithreading. The application has two sets of predefined words that have either positive or negative meaning:

positive – "positive", "good", "satisfactory", "nice", "convincing", "adequate", "splendid", "excellent", "top", "exciting", "marvelous", "great", "best", "creative", "fine", "awesome",

negative – "negative", "bad", "slow", "weak", "worst", "fair", "poor", "marginal", "unsatisfactory", "boring", "false", "disadvantages", "miss", "worse", "below", "bottom".

Furthermore, the application receives as its input a set of keywords whose context should be evaluated as either positive or negative. The goal is to evaluate the context of the whole article and subsequently a set of articles and documents in the portal in which the given keyword appears. Note that this may obviously change in time as the articles will change every day.

4.2 Testbed Environment

For the following experiments, the author used several computer nodes, each of which has Intel(R) Xeon(TM) CPU 2.80GHz processors with 4 physical processing cores with HyperThreading for a total of 8 logical processors, 2MB cache, 4GB RAM running Linux version 2.6.32-220.13.1.el6.x86_64 (mockbuild@c6b6.bsys.dev.centos.org) (gcc version 4.4.6 20110731 (Red Hat 4.4.6-3) (GCC)) SMP.

The nodes have access to a shared space mounted over NFS from another server with the same specifications. The network used is Gigabit Ethernet. The server is connected to an Dell/EMC AX150 disk array through FibreChannel. The disks are configured using RAID 5.

For controlling the number of cores used within a computer node, command `taskset` was used.

4.3 Real Parallel Experiments

In order to assess the performance of the solution, real tests were performed with a variable number of nodes and processors within nodes. This refers to either physical cores in multicore CPUs or cores available thanks to the Intel HyperThreading technology. Furthermore, the workflow application was run for input data sets with various: overall size of the files to be searched, number of files, average size of file, number of directories in the directory structure.

For the scalability results to be meaningful, the proposed solution needs to be tested on real documents. For the tests, the author used two sets of input data that was the New Technology (NT) portal of Interia at `http://nt.interia.pl`. The parameters of the sets showing various values of the aforementioned parameters are given in Table 1. Tool `wget` was used for download of the data with various depths (the `-l` parameter). Additionally, a set with fewer large files SET 3 was generated to measure the speed-up in the parallel system in which the overhead for reading many small files was minimized. Context of one keyword (a company name) was used in the experiments. For more keywords, obtained speed-ups would be even better due to a larger ratio of computations versus communication.

Name of data set	size [GB]	no of files	no of subdirectories	no of top level subdirs
`nt.interia.pl` SET 1	2.26	21398	258	40
`nt.interia.pl` SET 2	10.4	88537	1211	117 (5*)
large files SET 3	8.3	64	0	0

*after preprocessing for faster start-up

Table 1: Parameters of the data sets used for real tests

SET 1. 2.26 GB, 21398 files. Figure 3 presents the execution time of the application for data set SET 1 while Figure 4 shows corresponding speed-up values. The speed-up on 64 logical processors (32 of which result from the HT technology) is 26.3.

SET 2. 10.4 GB, 88537 files. Furthermore, the author increased the size of the problem by downloading the portal with a larger depth and obtained input data SET 2. This makes the problem larger, especially in terms of the number of files. Execution times and speed-up values are presented in Figures 5 and 6 respectively. The speed-up on 64 logical processors (32 of which result from the HT technology) increased slightly compared to SET1 to 27.7. One can observe that the following aspects limit the speed-up:

1. processing reasonably small files which results in reasonably high overhead for file management compared to search time,
2. HyperThreading which is not as effective as physical cores,
3. partitioning of initial data into subdirectories to be assigned for parallel computations.
4. not considering file sizes in the partitioning scheme. On one hand, this may lead to imbalance, on the other hand solving this problem for a large number of files of different sizes, even with heuristic algorithms may take too much time that might cancel savings on better load balancing.

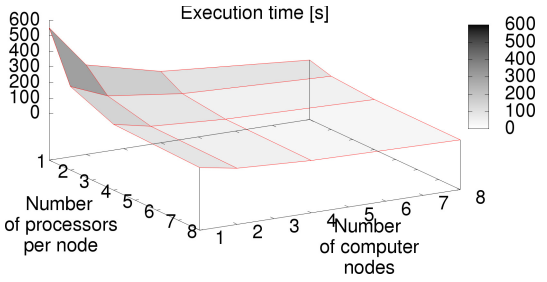

Figure 3: Execution time [s] vs number of computer nodes and processors per node for `nt.interia.pl` SET 1

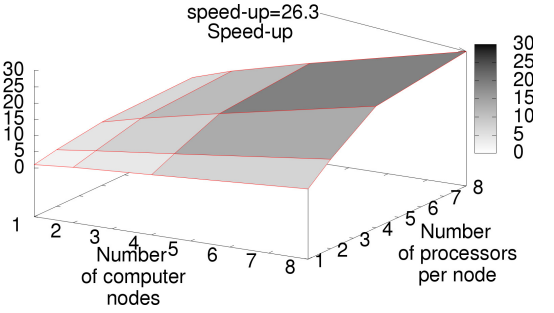

Figure 4: Speed-up vs number of computer nodes and processors per node for `nt.interia.pl` SET 1

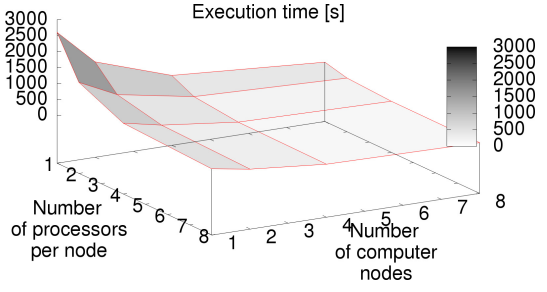

Figure 5: Execution time [s] vs number of computer nodes and processors per node for `nt.interia.pl` SET 2 with preprocessing for fewer (5) top level directories

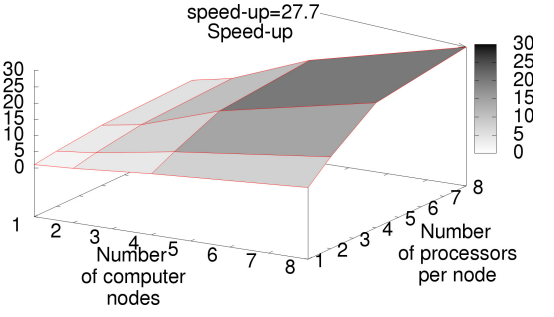

Figure 6: Speed-up vs number of computer nodes and processors per node for `nt.interia.pl` SET 2 with preprocessing for fewer (5) top level directories

Issues 1 and 2 are specific to input data and hardware respectively. For issue 3, listing and partitioning the files among the available nodes will take several seconds. In order to mitigate this problem, the following strategy is applied according to the scheme proposed in Section 3.2.1:

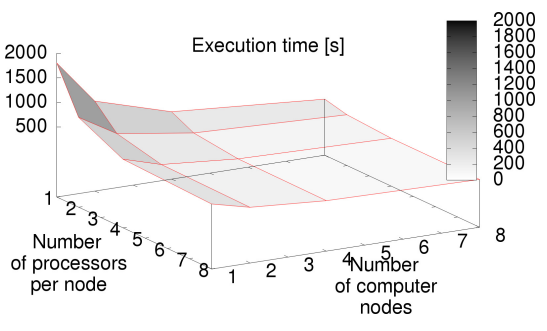

Figure 7: Execution time [s] vs number of computer nodes and processors per node for large files SET 3

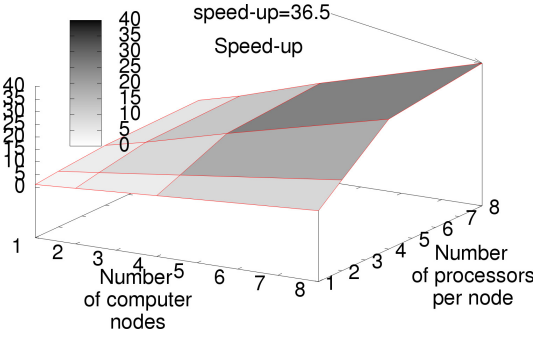

Figure 8: Speed-up vs number of computer nodes and processors per node for large files SET 3

1. partition the top level directory that includes many directories (117 for SET 2) into a few (5-10) directories. Five directories were used for SET 2 and these were generated by a script. This process is very quick, well below a second.

2. Operate in iterations on the few directories obtained in the previous step.

The goal of this improvement is to be able to operate with smaller data sets (directories) than the initial data set. This makes partitioning faster and consequently services and thus applications can start analysis sooner. The first step is equally important. Without it, the code would operate one by one on each of the initial 120 directories. While this would allow processes and consequently threads to start even faster that e.g. for 5 directories, each of the 120 directories is reasonably small. As a result, it would be more difficult to balance such small data sets across many processors. Because of that, it is best to create medium sized directories to find balance between the two. Real gains in execution time from this optimization are shown in Figure 9.

SET 3. 8.3 GB, 64 files of same sizes. As a benchmark for potentially better speed-up values, the author has also measured the scalability of the processing workflow applications for much larger files. This corresponds to searching in scientific applications such as: searching for occurrence of keywords and specific descriptions of the keywords in files generated by simulations in which certain events are marked by keywords. The parameters of this large files data SET 3 are listed in Table 1. Corresponding execution times and speed-up values are shown in Figures 7 and 8 respectively. The speed-up on 64 logical processors (32 of which result from the HT technology) is 36.5.

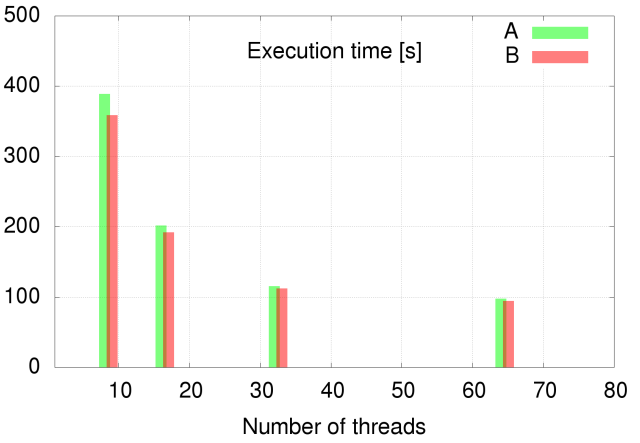

Figure 9: Comparison of execution times [s] for `nt.interia.pl` SET 2 without (A) and with (B) preprocessing for fewer (5) top level directories

4.4 Discussion

Based on the measured results, the author performed analysis in order to determine the factors limiting speed-up of the workflow application. Figure 10 presents the following speed-ups:

1. ideal without communication and assuming up to 64 physical processors/cores. This does not correspond to the testbed environment as the number of physical cores is smaller.

2. ideal without communication and assuming 32 real cores and 32 HyperThreading cores. This corresponds to the testbed environment but without imbalance, synchronization etc. The value for 64 processors was computed taking into account relative performance for the application tested on real vs HyperThreading cores.

3. real speed-up for SET3. In this case, the data is quickly partitioned equally among the processes and then threads. This case can serve as an upper bound for any input data. The drop in performance comes from bottlenecks in memory and disk access.

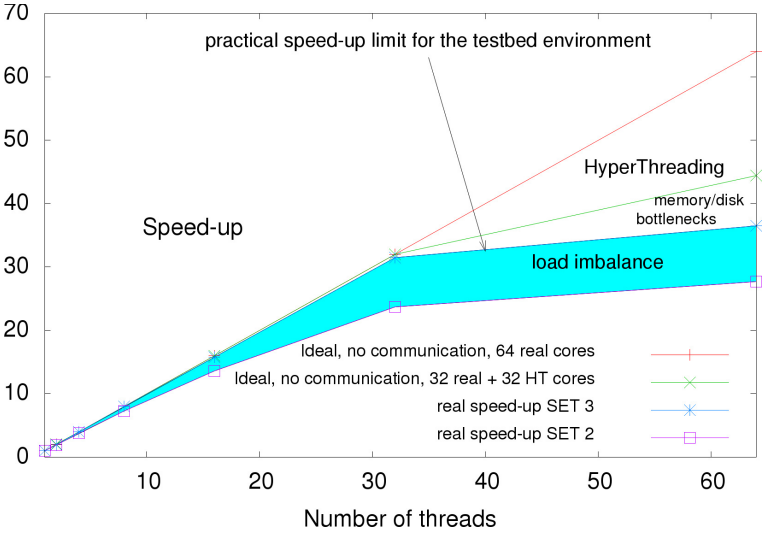

Figure 10: Factors limiting speed-up

4. real speed-up for SET2. In this case, compared to SET3, lower speed-up comes from imbalance of data among processes and then threads. Although there is a large number of files, assuming the same number of files per process does not balance the data perfectly. On the other hand, better partitioning would require consideration of file sizes and would take several seconds to complete for the tested numbers of files shown in Table 1. Given that the total time on 64 threads is around 90 seconds for SET3, better balance might not cancel the overhead for better scheduling. This is considered as a field for future research.

5 Conclusions and Future Work

The paper presented a workflow application for parallel processing of data sets presumably downloaded from the Internet. Parallel processing is performed at two levels: several computers in parallel and on multiple processors/cores within each node. Optimization through partitioning of data allowing fast startup of processing was presented. Concrete services were implemented for assessment of whether the context of given keywords in the given data set is positive or negative. The data set downloaded from the Internet is stored as a collection of files in a directory structure. Depending on the sizes and characteristics of the real data sets from a new technology portal, execution times and corresponding speed-ups in the range between 26.3 and 36.5 were obtained on 64 cores, 32 of which were available thanks to the HyperThreading technology. Factors limiting the speed-up were discussed with impact on performance.

In the future, the author plans extending the solution to other systems including processing on GPU cards as well as Intel Xeon Phi technologies. Additionally, trade-offs between more accurate load balancing and its cost will be investigated in the context of the output speed-up.

References

[1] David P. Anderson. Boinc: A system for public-resource computing and storage. In *Proceedings of 5th IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, USA, November 2004.

[2] J. Balicki, H. Krawczyk, and E. Nawarecki, editors. *Grid and Volunteer Computing*. Gdansk University of Technology, Faculty of Electronics, Telecommunication and Informatics Press, Gdansk, 2012. ISBN: 978-83-60779-17-0.

[3] Rajkumar Buyya, editor. *High Performance Cluster Computing, Programming and Applications*. Prentice Hall, 1999.

[4] Stefan Büttcher, Charles L. A. Clarke, and Gordon V. Cormack. *Information Retrieval*. MIT Press, 2010.

[5] Sang-Hwa Chung, Soo-Cheol Oh, Kwang Ryel Ryu, and Soo-Hee Park. Parallel information retrieval on a distributed memory multiprocessor system. In *Algorithms and Architectures for Parallel Processing, 1997. ICAPP 97., 1997 3rd International Conference on*, pages 163–176, 1997.

[6] Pawel Czarnul. Integration of compute-intensive tasks into scientific workflows in beesycluster. In *Proceedings of ICCS 2006 Conference,*, University of Reading, UK, May 2006. Springer Verlag. Lecture Notes in Computer Science, LNCS 3993.

[7] Pawel Czarnul. A model, design, and implementation of an efficient multithreaded workflow execution engine with data streaming, caching, and storage constraints. *The Journal of Supercomputing*, 63(3):919–945, 2013.

[8] Pawel Czarnul. Modeling, run-time optimization and execution of distributed workflow applications in the jee-based beesycluster environment. *The Journal of Supercomputing*, 63(1):46–71, 2013.

[9] Pawel Czarnul and Krzysztof Grzeda. Parallelization of electrophysiological phenomena in myocardium on large 32 & 64-bit linux clusters. In Springer-Verlag, editor, *Proceedings of Euro PVM/MPI 2004*, volume LNCS 3241, pages 234–241, Budapest, Hungary, Sept. 2004.

[10] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[11] Xuefeng Guan. A robust parallel framework for massive spatial data processing on high performance clusters. In *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, XXII ISPRS Congress*, volume XXXIX-B4, Melbourne, Australia, August 2012.

[12] Tawa Khampachua. Parallel information retrieval on a pc-cluster using vector space model. Software Engineering Laboratory, Department of Computer Engineering Chulalongkorn University, `http://www.cp.eng.chula.ac.th/~piak/teaching/seminar/phd2001/tawa_slide.pdf`.

[13] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. Parallel data processing with mapreduce: a survey. *SIGMOD Rec.*, 40(4):11–20, January 2012.

[14] Zhihong Lu, Kathryn S. McKinley, and Brendon Cahoon. A performance evaluation of parallel information retrieval on symmetrical multiprocessors. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.5016.

[15] P.D. Michailidis and K.G. Margaritis. Parallel text searching application on a heterogeneous cluster of workstations. In *Parallel Processing Workshops, 2001. International Conference on*, pages 169–175, 2001.

[16] Gerard Salton and Chris Buckley. Parallel text search methods. *Commun. ACM*, 31(2):202–215, 1988.

[17] Julian Szymański. Words context analysis for improvement of information retrieval. In *Proceedings of 4th International Conference on Computational Collective Intelligence. Technologies and Applications*, pages 318–325. Springer, Lecture Notes in Artificial Intelligence, 2012.

[18] Daniel Warneke and Odej Kao. Nephele: efficient parallel data processing in the cloud. In *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, MTAGS '09, pages 8:1–8:10, New York, NY, USA, 2009. ACM.